An Empirical Analysis of XML parsing using various operating systems.

Amitesh Saxena, Dr. Snehlata Kothari

***Abstract*— As the use of internet technologies are widely increasing, the XML markup language attains a remarkable importance due to its language neutrality and independency in using data exchange and data transfer through web environment mechanism. For improving the processing performance of XML parser, it is necessary to find out a mechanism, in which we get minimum processing time while parsing of XML documents.**

In this paper, XML documents are being experimentally tested using various operating systems to determine, whether an operating system effect the processing time of XML parsing.

***Index Terms*— XML Parser, DOM Parser, Operating system.**

I. INTRODUCTION

In Present world, mega information are sharing and transmitting, In this XML plays a very significant role as a worldwide design for data interchange. It allows users to share XML documents. XML is capable to the mining of data from an XML document without any facts or knowledge about the contents of that. XML documents need to be conformant with XML specifications for achieving this transparency. By using an XML parser, this specification conformance can be checked. The parser makes the data easy to get and also ensures the validity of that.

In today's, a large number of XML parsers, coded in a verity of languages, May be these parser not give the similar performance in terms of parsing speeds, accuracies, and storage requirements. This is lastly proofed that, for execution time savings, accurate parsing, and storage requirements, a parser must be selected to fit those specific requirements.

In this study, XML document will be tested with three DOM API, i.e. PHP,JAVA and Microsoft on various Operating Systems like WIN7, WIN8, UBUNTU, Red Hat, etc. The main objective of this study is to check whether the operating system affects the parsing speed and if yes then the best combination of parser and operating system will be also finding out.

II. TECHNOLOGY OVERVIEW

In this section we describe about different XML Parsers.

A. XML Parser

A parser is a piece of program that takes a physical representation of some data and converts it into an in-memory form for the program as a whole to use. Parsers are used everywhere in software. An XML Parser is a parser that is designed to read XML and create a way for programs to use XML. There are different types, and each has its advantages. Unless a program simply and blindly copies the whole XML file as a unit, every program must implement or call on an XML parser.

Amitesh Saxena, Research Scholar of Pacific University, Udaipur
Dr. Snehlata Kothari, Pacific University, Udaipur.

B. DOM (Document Object Model)

It supports navigating and modifying XML documents
- Hierarchical tree representation of document
- Tree follows standard API
- Creating tree is vendor specific
DOM is a language-neutral specification
- Binding exist for Java, C++, CORBA, JavaScript, C#
- can switch to other language.
The Document Object Model (DOM) is an interface-oriented application programming interface that allows for navigation of the entire document as if it were a tree of node objects representing the document's contents. A DOM document can be created by a parser, or can be generated manually by users (with limitations).

III. LITERATURE REVIEW

Review of work already done on the subject

A study of various XML parsers could be very positive in determining the strengths and weaknesses of various XML parsers in respect to the different features of XML. Parsers can be compared to one another by checking their conformance with the XML recommendations [1] given by the World Wide Web Consortium. The Organization for Advancement of Structured Information Systems (OASIS) [2] is a NGO that has collected a number of test cases from various sources and built a Conformance Test Suite for XML with approx. 2000 test cases (as of November 6, 2001).

Anez [3]. 1999, conducted a study to determine the suitability of XML and Java for the representation and manipulation of Transport and Land Use (TLU) modeling information as used in urban and regional planning. This study evaluated seven different XML parsers with respect to conformance with XML specifications, speed and memory usage. In this study, the test suite developed by James Clark (currently part of OASIS Test Suite) was used for checking the conformance of different parsers. Speed and memory usage tests were performed using two large XML files (0.8 and 1.2 MB, respectively). Each of these two files contained several thousand XML elements nested in a four-level deep hierarchy, and all of the elements had one or more attributes. The study concludes by giving rankings to the different parsers as shown in Table 1.

Table 1: Rankings for different parsers by Anez [3]

Parser	Rank
IBM XML4J (XML for java) v 1.1.4	Outstanding
James Clark's XP v0.4	Good
Microsoft XML (MSXML) v 1.9	Good
Microstar Aelfred v1.1	Good
Sun XML (under construction)	Acceptable
Loria sxp v 0.72	Acceptable
Data Channel XML parser	Poor

This study also compares the relative performances of the parsers with respect to conformance with XML recommendations, speed, and memory usage. It concludes that different parsers excel under different requirements. However, this study does not provide quantitative data about the conclusions.

Claben [4]. 1999, considered the following parsers: IBM XML4J, Apache Xerces, Sun Project X, Microsoft MSXML, Oracle XML parser for Java, and James Clark XP. He used the following features for comparing the different parsers: Well-formedness, Validity, XML Schema, Namespaces, XSL-T, SAX levels 1, 2, and DOM levels 1, 2. The focus of this study was to determine the set of features supported by each of the parsers using the OASIS Conformance Test Suite (1000 test cases1). The study concludes that Sun's parser and James Clark XP are the best parsers supporting the XML standard, but it does not give any quantitative figures of how good or bad each parser is with respect to a specific feature.

Cooper [5], 1999 studied how parsing speeds vary with the programming language used for developing the parser. In this study, two java parsers, two C parsers, one perl and one python parser were used. Five XML documents with sizes ranging from 160 K to 5.0 MB were used in this study. This study concluded that C parsers are always faster compared to java, perl or python parsers.

Mohseni [6]. 2001 performance test indicates that MSXML rivals other parser having the shortest loading time.

Noga, M., Schott, S., L¨owe, W. [8]. 2002, Therefore efforts have been made to improve DOM parser performance by exploiting lazy XML parsing. The key idea is to avoid loading unnecessary portion of the XML document into the DOM tree. It consists of two stages. The pre-parsing stage builds a virtual DOM tree and the progressive parsing stage expands the virtual tree with concrete contents when they are needed by the application.

Oren [9]. 2003, proposes Piccolo XML parser presenting a comparative study between parsers, which implements SAX (*Simple API for XML Processing*) 5 interfaces. Although outdated, this study provided interesting guidelines related to the test methodology and conclusions about the overall best API, which changes in subsequent studies for similar tests.

Van Engelen et al. [12]. 2005, uses deterministic finite state automata (DFA) to integrate them and the DFA is built upon the schema according to mapping rules.

Takase et al. [13]. 2005, explores a different way to improve parser performance. It memorizes parsed XML documents as byte sequences and reuses previous parsing results when the byte sequence of a new XML document partially matches the memorized sequences.

Sosnoski [14]. 2005, carried out a test on DOM based parsers using XMLBench. He tested on the execution speed and memory usage for a set of XML documents ranging from small-scale to large-scale file sizes. The test result shows that Xerces outperforms among the others. Besides, Xerces parser is also voted as the best XML parser of the year by XML-Journal/Web Services Journal Readers' Choice Awards [15-2004].Since Xerces and MSXML outperform the rest of the parsers in most cases, we have decided to concentrate benchmarking our proposed parser, xParser against these two parsers.

Perksins et al. [16]. 2005, where authors use a small (less than 1 KB) XML representing a typical purchase order structure to test transcoding impact and object creation of DOM, SAX and JAX-RPC. The authors also explore the navigation costs of each API and compare the results with a specific XPath parser.

A study towards different XML parsers is beneficial when comes to determine the strength and weaknesses of the products. Various studies have been conducted which compare on conformance to standards, speed, memory usage and so on.

Lam T.C., Ding J.J. and Liu J.C. [20]. 2008, concluded that the process of handling XML documents was described in four phases: *Parsing*, that is considered a critical step in performance, *Access*, *Modification* and *Serialization* (figure 1), whose performance is directly affected by the parsing models.

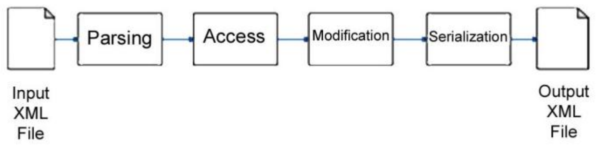


Figure 1. Example of a XML memory tree representation

As the most critical factor of performance, parsing is characterized by the conversion of characters, mainly related to the conversion of characters into a format that a programming language understands, lexical analysis which is the process that identifies XML elements, e.g. start node, end node or characters, applying regular expressions defined by World Wide Web Consortium (W3C)$_1$. The last step of the parsing phase is the syntactic analysis of the document, where it is checked if the document complies with the rules of construction of an XML document. Finally, the API implements access and modification operations on the data resulted from the parsing process.

They analyzed parsing models, data representations and their impact on XML processing. They concluded that both DOM and VTD are good for back-and-forth data access. VTD parses faster than DOM and consumes less memory. VTD is better for simple and rare modifications, while DOM is better for complex and frequent ones. SAX and StAX are appropriate for applications with extremely restrictive memory but not for back-and-forth access or modification. In a nutshell, DOM is most suitable for database applications, while SAX and StAX are more appropriate for streaming applications. VTD is a good candidate for hardware acceleration based on its symmetric array structure, but its effectiveness in real-world applications using a commercial hardware accelerator remains an open question

Due to its complexity and importance, the parsing process is the most critical operation in XML processing, directly conditioning processing time and memory consumption. Several studies have been conducted with the goal to test, improve representation models and APIs in XML processing. http://www.w3.org/

As Java and other technologies evolve, it is necessary to review the new approaches and improvements provided by several XML parsers available.

Chengkai Li [21]. 2009 concluded in his research that every XML application has to parse an XML document before it can access the information in the document and perform further processing. Therefore XML parsing is a critical component in XML applications.

DOM is memory intensive since it has to hold the entire document tree in memory, making it incapable in handling very large documents.

Sankar, P.Krishna and Shangaranarayanee, N.P. [23]. 2011, Concluded that Java API for XML Processing (JAXP), which process XML documents by using, the Document Object Model (DOM) method, the Simple API for XML (SAX) method, and Streaming API for XML (StAX) method are used commonly. As StAX name indicates, it is targeted at streaming applications such as the merging of two documents and exchange information between cooperating entities. StAX allows an application to process multiple XML sources simultaneously. Among the DOM and SAX widely used methods, StAX provides the parsing efficiency and making developer comfort.

SAX

SAX stands for Simple API for XML. Its main characteristic is that as it reads each unit of XML, it creates an event that the calling program can use. This allows the calling program to ignore the bits it doesn't care about, and just keep or use what it likes. The disadvantage is that the calling program must keep track of everything it might ever need. SAX is often used in certain high-performance applications or areas where the size of the XML might exceed the memory available to the running program. The design inspiration and subsequent coordination was done by Dave Megginson, who continues to maintain the SAX Project website. The SAX standard currently is at version 2.0.

SAX is a push parser, since it pushes events out to the calling application. Pull parsers, on the other hand, sit and wait for the application to come calling. They ask for the next available event, and the application basically loops until it runs out of XML.

A. StAX — Streaming API for XML

The StAX pull-parser has been implemented in the Java world by a standard called JSR-173. Both Saxon and Data Direct XQuery support pull parsing. In some instances, particularly in Data Direct's implementation, pull parsing can give a significant performance boost, but both implementations have been so highly tuned that the choice between SAX, DOM and StAX for any given application is a matter for testing. Since within Stylus Studio® XML Enterprise Suite the XML Pipeline constructor knows the capabilities of each node in the pipeline, this choice is handled automatically for you.

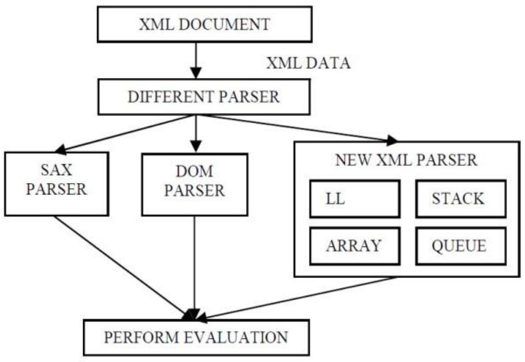


Figure 2. Architecture View

Traditionally SAX and DOM parsers are use to parse the XML document. DOM creates tree structure of whole XML document in main memory and parses the document. Unfortunately, DOM has some penalty over performance characteristics. This method involves reading the entire file and storing it in a tree structure, which may be inefficient, slow, and it can be a strain on resources. One alternative is SAX. SAX allows you to process a document as its being read, which avoids the need to wait for all of it to be stored before taking action. SAX generates events by fetching contents from secondary storage during parsing and unfortunately secondary storage is slower. Data structure based parser works in main memory and uses various data structure for parsing. In the implementation, the proposed parser removes the elements from document and serially checks if the document is well formed or not using Linked list, Queue, Stack and Array simultaneously, which increases its performance over SAX and DOM parser.

I. RESEARCH WORK

Research gaps identified in the proposed field of investigation:

All available Studies have concluded that a significant portion of their execution time is being spent in XML data processing [26], mainly in XML data parsing. The role of data parsing is to convert the input XML document and divide it into small elements. It is mainly a significant portion in XML data processing because an XML document must be parsed before any other operations can be executed. Studies have shown that data parsing consumes about 30% of web service applications [27].

All comparative studies have done in keeping mind about Parser, means all last compares are performed in context of just Parsing API. No studies have done in the context of mechanism or environment or platform which may effect on time consumption in Parsing of XML document. There are no study encountered, which find out the impact of operating system on parsing of XML document.

So there is gap identified that, does an operating system effects the parsing time of XML document?

Main points included in research are

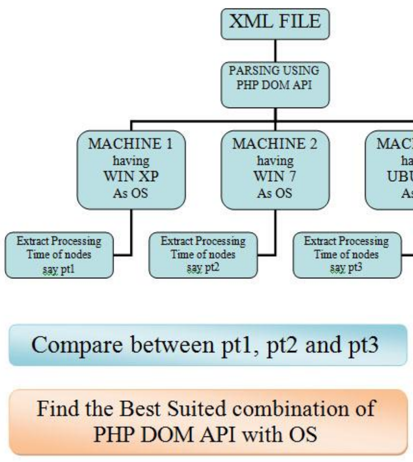
RQ1- Whether the Operating System effects the processing of XML DOM parsing?

RQ2- Which OS suit to XML DOM API for giving the better performance?

RQ3- Whether the processing performance may be improving in context of OS or not?

IV. Research Design

We used Descriptive statistics along with 1x3 factorial ANOVA Technique and for the comparison mean, SD, z-test, t- test have been performed for data analysis.


XML FILE
PARSING USING PHP DOM API
MACHINE 1 having WIN XP As OS
MACHINE 2 having WIN 7 As OS
MACHINE 3 having UBUNTU As OS
Extract Processing Time of nodes say pt1
Extract Processing Time of nodes say pt2
Extract Processing Time of nodes say pt3
Compare between pt1, pt2 and pt3
Find the Best Suited combination of PHP DOM API with OS

V. Future work :

In this research we will experimentally compare parsing performance of one XML API, i.e. PHP DOM API by using three different operating system, in future we will compare the different XML API of different companies (in before different APIs are compared, but of same company like JDOM, SAX, STAX) like PHP DOM, JDOM, MS DOM, etc, by either using one operating system or three operating systems.

VI. Conclusion

We are working on the parsing technique to find the best parsing technique for different operating system. We have displayed the working modal of my research. We use Descriptive statistics along with 1x3 factorial ANOVA Technique and for the comparison mean, SD, z-test, t- test have been performed for data analysis.

References:-

[1] Extensible Markup Language, http://www.w3.org/TR/REC-xml.

[2] OASIS, http://www.oasis-open.org.

[3] Juancarlo Anez, "Java XML Parsers-A Comparative Evaluation of 7 Free Tools," Java Report Online, February 1999.

[4] Michael Claben, XML Parser Comparison, http://www.webreference.com/xml/column22/index.html. Feb 1999.

[5] Clark Cooper, Summary of XML Parser Performance Testing http://www.xml.com/lpt/a/Benchmark/exec.html. May 05, 1999.

[6] Mohseni, P., “Choose Your Java XML Parser”, 2001, http://www.devx.com/xml/Article/16921.

[7] Karre, S. and Elbaum, S., “An Empirical Assessment of XML Parsers”, *6th* Workshop on Web Engineering, 2002, pp. 39-46.

[8] Noga, M., Schott, S., L¨owe, W. (2002): Lazy XML Processing. In ACM DocEng, ACM Press, New York, 2002.

[9] Y. Oren, “SAX Parser Benchmarks”, *http://piccolo. sourceforge.net/bench.html*, 2002.

[10] Nicola M. and John J. (2003): XML parsing: a threat to database performance. CIKM 2003: 175-178.

[11] Elliotte, R.H., “SAX Conformance Testing”, XML Europe, 2004.

[12] Van Engelen, R. (2004): Constructing finite state automata for high performance XML web services. In Proceedings of the International Symposium on Web Services (ISWS), 2004.

[13] Takase T., Miyashita H., Suzumura T., and Tatsubori M. (2005): An adaptive, fast, and safe XML parser based on byte sequences memorization. WWW 2005: 692-701.

[14] Sosnoski, D.M., “XMLBench”, 2005 http://www.sosnoski.com/opensrc/xmlbench.

[15] XMLJ News Desk, “Journal Readers choice Award”, 2004 http://xml.sys-con.com/read/44008.htm.

[16] E. Perkins, M. Kostoulas, A. Heifets, M. Matsa, and N. Mendelsohn, “Performance Analysis of XML APIs”, in *XML 2005 Conference proceeding*, 2005.

[17] Kostoulas M., Matsa M., Mendelsohn N., Perkins E., Heifets A., and Mercaldi M. (2006): XML screamer: an integrated approach to high performance XML parsing, validation and deserialization. WWW 2006: 93-102.

[18] Farf´an F., Hristidis V., and Rangaswami R. (2007): Beyond Lazy XML Parsing. DEXA 2007: 75-86.